Internal Report 2010–16 September 2010

Universiteit Leiden
Opleiding Informatica

Bundle Tracing

using

Geometric Algebra

Simon Zaaijer

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Bundle tracing using Geometric Algebra

Simon Zaaijer
Supervisor: André Deutz

September 9, 2010

Contents

Introduction

Bundles are sets of rays between triangular plane segments, which are used as primitives for tracing light in the technique described here. By tracing bundles instead of separate rays, the geometric information of the scene can be used more efficiently, as will become apparent in section 2.4. From these bundles, individual rays can be used to compute the light intensity. The information that is stored with the bundles is sufficient for knowing which of these rays should be omitted due to obstruction by other plane segments. While the geometric computation is done through bundles, the shading function is left to be computed for the separately generated rays.

1 Tracing light

In geometric optics (see [1]), light is treated as a wave motion where the wavelength is considered negligible compared to the dimensions of the relevant components of the optical system. In this model, light travels from one point to another along straight lines called rays. The path followed by the light is influenced by the media through which it travels. A ray may be reflected, refracted and absorbed on the interface dividing two media. In computer graphics, this interface is commonly represented by a set of triangular plane segment, known as faces, present in the scene, the environment that is to be rendered. The laws for reflection and refraction as provided by geometric optics can easily be applied to rays colliding with such planes.

Tracing these light rays allows us to determine the amount of light reaching the viewpoint. By mimicking the effects of a digital camera, the intensity of the light allows us to generate an image of the scene. Since the scene may contain many light sources, but will only contain one viewpoint, tracing the light from the viewpoint ensures each path of light is followed just once. When tracing from the light sources separately, the same path to the viewpoint will be traced multiple times. For this reason, it is convenient to trace the rays in the opposite direction, from the viewpoint to the light sources. As geometrical optics states, the path followed by a ray in a reverse direction, is identical to that of the original ray. In other words, the same laws apply when tracing light from the viewpoint to the light source.

1.1 Ray tracing and Rasterization

Ray tracing and rasterization are the two main techniques used in computer graphics. (see chapters 9 and 12 of [3]) Rasterization is currently mainly

used in games and animation movies. While rasterization is a lot faster, allowing it to run in real time with any graphics card, it lacks in quality and does not provide a physically correct result. Ray tracing as a recursive global illumination technique, known as Path tracing, can provide physically correct results however, but to render a scene realistically takes far too long to be practically useful. For now, ray tracing is therefore mainly used for single rendered images, although local illumination ray tracing with global illumination approximation can now be done in real time.

The main difference between the two approaches is that rasterization attempts to draw the triangles of the scene onto the output image by performing a projection onto the view plane, while ray tracing generates rays from the view plane, traces them separately and maps their intensity to the output image.

Bundle tracing borrows the principles of both techniques. It combines the per triangle approach of rasterization to quickly generate only the required rays of ray tracing. Through this combination, bundle tracing retains the flexibility of modern ray tracing variants, while accelerating and optimizing the process through the use of geometry, somewhat like a modern rasterizer.

1.2 Triangulation and the Scene

In almost every modern application of computer graphics, the scene, the environment to be rendered, is triangulated, so that it is easy to use by the rendering algorithm. Especially rasterization requires all of the geometry to be formed of triangles. Ray tracing techniques do not have this requirement, but for speed considerations and practical use, these techniques also commonly use geometry existing largely or solely of triangles. There are two main reasons for this. The first is that triangles are easy to store on a computer. Only the three points at the corners, known as the vertices, need to be stored. Other information, such as normal vectors or texture coordinates, is optional. The second reason is that it eases the computation. Using only triangles makes the scene a lot less complex, effectively making computations far less intensive, but it may require more memory compared to using rounded shapes, such as bézier surfaces.

Naturally, triangulation limits the shape of geometry. To make every triangle small enough for it to be indistinguishable would provide a solution for this limitation, but that would defy both of the reasons for triangulation. Storing such a large amount of triangles is difficult and using them in any computation would be very costly. Instead, several techniques are used to make a triangle look more complex than it really is and, when used correctly, they can allow a scene to seem realistic with a minimal amount of geometric

complexity.

Texture mapping for instance applies an image of a material to a triangle, which greatly increases realism compared to using monochrome triangles. Bump mapping adds a fake sense of depth in a triangle, where the lighting or texture placement is adjusted so that parts of the triangle seem to come forward or fall back. A last very common technique is the interpolation and averaging of normals on each vertex. Instead of just computing the normal for each face, the normals of the faces surrounding a vertex are averaged, so that the resulting surface appears smooth rather than blocky.

Besides these triangles, the scene generally includes one viewpoint and one or more light sources, both with their own properties for orientation and position. Light sources typically have an intensity property as a measure for the amount of light they radiate. These light sources are typically points or small spheres, either omni directional, directional (such as sunlight), or as a spot targeting a point.

2 Bundles of rays

Instead of computing with separate rays, the method we describe here handles light in bundles of rays. Through these bundles, the light is traced from the viewpoint to the light source. This bundle is not simply a cone of rays, but consists of a more freely defined set of rays, something that will be explained in more detail later on.

2.1 Introducing bundles

To get an idea of what a bundle is and how they interact with the environment, let us start by looking at an example. To visualize this more simply, the example given here is in 2D. The description will specify how the given elements can be extended to 3D. In contrast to how light is traced in the bundle tracer, we will start here at a light source, since tracing light in its physical direction should be more intuitive.

Let us start by looking at a simple scene with some line segments (Figure 1). Although drawn as lines, these segments should be seen as planes, faced perpendicular to the viewer.

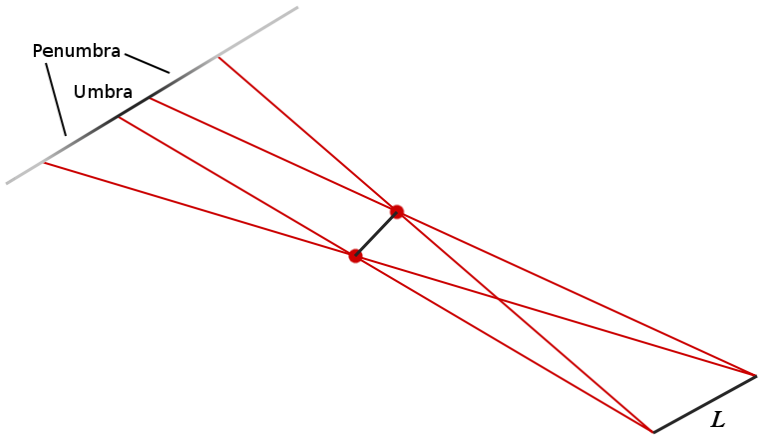

Figure 1: A light source L emits light onto an object, which casts a shadow on a distant line.

When creating a bundle at the light source L, it will initially contain all possible rays leaving the line from one side. Without any further information, this means that anything on that side of the line may be lit. After leaving its source, the light will first reach another line, the shadow casting object in

the middle. It is important to regard this line first, since the effects this line will have on the bundle are of consequence to the light reaching the distant line. In this case, the effect results in a shadow.

When the bundle hits the shadow casting line, some of its rays actually hit the line, while others will pass by it. To effect this, the bundle is split into 3 *subbundles*. One will hit the line and two others will pass by it on either side. The bundle hitting the line has now become *confined*. This means it will only contain rays that lead to the target line, the shadow casting object.

The two bundles passing by the line are also restricted, but in a different way. The only restriction these are given has to do with the end points of the line segment, shown in Figure 1 as two red dots. Only the rays passing on one side of each dot are considered part of the bundle.

To scale this idea to 3D, it is best to imagine the dots as line segments perpendicular to the paper. One of these moves down into the paper, while the other moves up, so that their orientations are opposite. Because of their opposite orientation, the side that allows light to pass through will also be opposite, thus allowing light to pass on either side of the line, but not through it. These dots, lines in 3D, are called *cutting edges*, as they seem to cut into the bundle from one side.

The effect of this restriction can be seen on the distant line as the shadow cast by the middle object. Notice how a part of the distant line is completely in shadow. Next to that are two parts which still receive some light, but not from the entire light source, due to the restricting cutting edges. This results in a smoothly fading *penumbra*. The sides of the line are completely lit.

When the two passing subbundles hit the distant line, they will also be split into subbundles. In this case, each of the two is split into a bundle hitting the line and a bundle passing by it. The bundle hitting the line will become confined, like the bundle hitting the shadow casting line. It will retain its cutting edges, as these are used to describe the shadow and penumbra. When passing by the distant line, the subbundle will receive another cutting edge from the end point of the line. Since this cutting edge cuts away the effect of the previous cutting edge completely, the previous one may be discarded. Note that discarding a previous cutting edge will not always occur and that it is a lot more likely to occur in a 2D space than in 3D. In 3D, bundles will often have many cutting edges as they pass more planes.

A slightly more complex scene is given in Figure 2. Here we can see what happens after a bundle hits a line or, in other words, after it has become confined. This image shows the paths a bundle and its subbundles will travel through this scene, where the number of reflections is limited to two.

In this scene, a shiny line S will reflect another object O, which is also directly visible and should thus show up twice from the camera. S will also

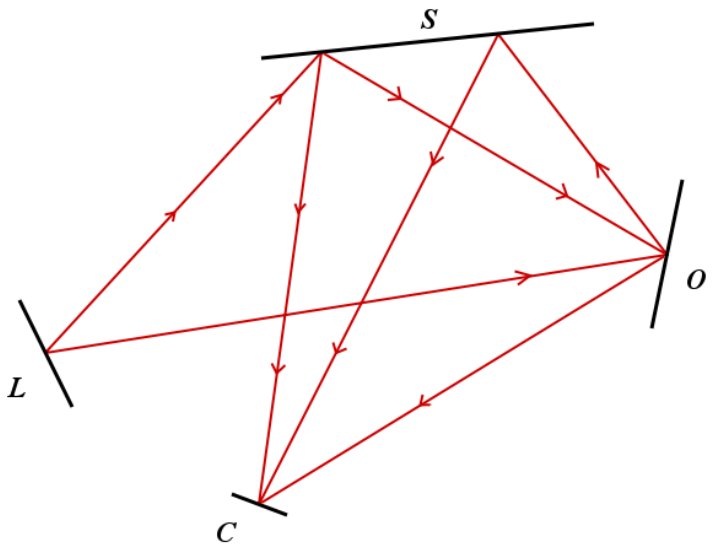

Figure 2: A light source L emits light onto an object O and a shiny object S. Several possible paths are shown for the light to follow. The light is caught by a camera C.

be reflected in O, but since O is more diffuse, it will not show up as clearly. The bundle from the light source L may first hit either the shiny line S or the other line O. These planes will not have any effect on the bundle leading to O, so they can be treated separately.

Once a bundle has reached a line, several things may happen, including reflection, refraction, absorption or a simple pass through. In this example we will look at the most prominent of these, reflection. Reflection will create a new bundle starting from the target plane. This new reflected bundle is a child of the old bundle. In this example, the amount of reflections is limited to two, which leaves us in the situation shown by the directed lines in Figure 2.

To show the relation between parent and child bundles, a simple graph of the example is given in Figure 3. The edges in this graph represent the bundles and the nodes represent the objects in the scene between which the bundles are created. Note that each leaf represents the camera object and the light source only appears at the top of the tree. When using bundle tracing, where the light is traced from the camera to the light source, this would be reversed.

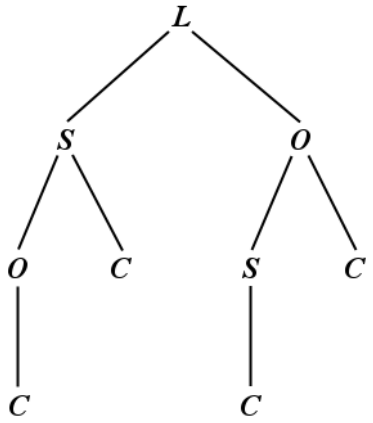

Figure 3: The different paths of light are drawn as a graph, where the edges are the bundles and the nodes are the objects in the scene (see also Figure 2).

2.2 Generic bundles

Most bundles are commonly formed on a triangular plane segment, triangle for short, by for instance a reflection of a bundle from the camera. The plane supporting the triangle from which a bundle originates will be called the *source plane* and the triangle it supports will be called the *source triangle*. From it, various rays of light can travel from any point on the triangular segment in any direction, although only the rays emanating from one side of the triangle are considered. In short, a generic bundle starting at a triangular segment contains all rays on one side of the source plane. This is of course far too much to be practically useful.

For this reason, we also define a *target plane* supporting a triangular segment, the *target triangle*, for a bundle. If a bundle has a target triangle defined, the bundle is called *confined*, as opposed to *unconfined* bundles where the target plane is not yet specified. A confined bundle consists of all rays traveling from one triangular segment to the other. This means that for each point on the source triangle a ray can be followed to each point on the target triangle. An unconfined bundle consists only of rays which have no end endpoint, so that they can move on to infinity.

Definition 2.1. A generic bundle is composed of a source triangle, based on a source plane, and a possibly empty set of cutting edges, where each cutting edge is a line. It contains all rays with the following restrictions:

- Each ray must start on the source triangle.

- The direction must face out of the front side of the plane, so that $n \cdot d > 0$, where n is the normal vector of the source plane and d is the direction vector of the ray.
- The ray must pass by the positive side of each cutting edge, so that, with a starting point p_1 and a line Λ, the cutting edge, for each point p_2 on the ray, $(\Lambda \wedge p_1)^* \cdot p_2 > 0$.

A generic bundle may either be confined or unconfined. If it is confined, the bundle will also contain a target triangle, based on a target plane. This results in another restriction:

- Each ray must have an endpoint on the target triangle.

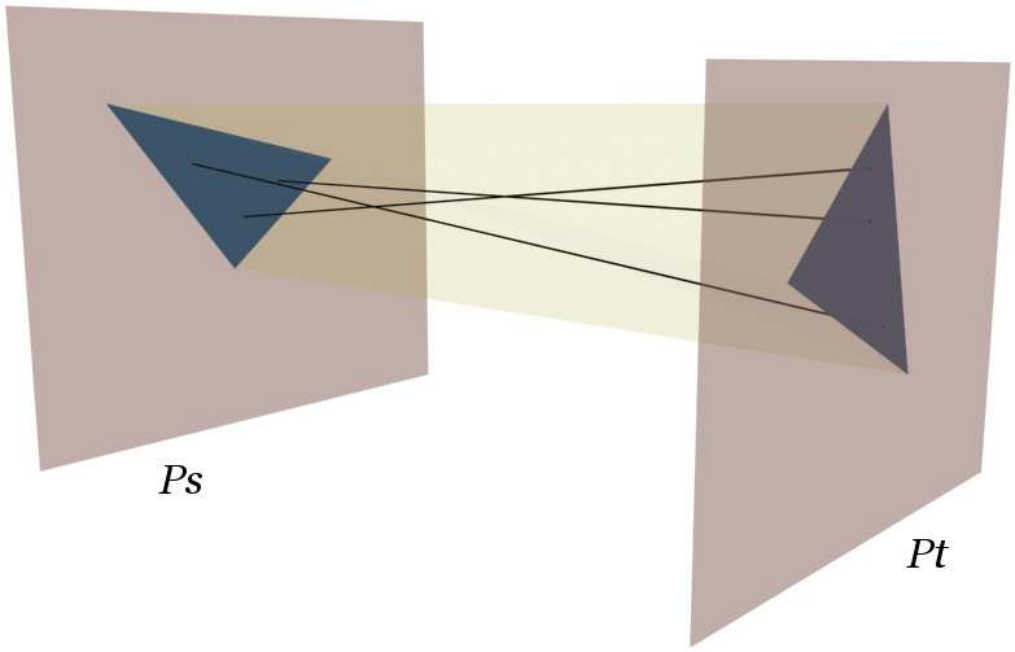

Figure 4: A generic confined bundle with source plane Ps and target plane Pt, with 3 exemplary rays drawn inside.

2.3 The camera and light sources

The bundle from the camera can be described as a single unconfined bundle starting at the lens. Of course the light passing through this bundle does not stop at the lens, but continues through it to the sensor plane. The refraction caused by the lens is modeled separately, such that each ray of the bundle from the lens hits the sensor plane at some point. This is done by placing a second bundle between the sensor plane and the lens. The bundle from the lens to the scene is called the *lens bundle* and the bundle between the sensor plane and the lens is called the *sensor bundle*.

Both of these bundles are exceptional. The lens is composed of multiple triangles, often shaped as a hexagon or octagon, due to the shape of the aperture. The curvature of the lens can be neglected, since most of its effects are noticeable through the computed refraction and the actual added depth is negligible compared to the size of the scene. The number of source triangles used for the lens requires a separately defined bundle.

Definition 2.2. The lens bundle is a generic bundle, where the source triangle is replaced by a set of source triangles, all based on the source plane and oriented equally.

The bundle between the sensorplane and the lens, the sensor bundle, is a very simple form of a bundle. It has a source plane and a target plane, which are always parallel. On the source plane is a rectangular plane segment, also knowns as a quad, which is the sensor plane. The target plane consists of the same triangles as the source triangles of the lens bundle.

Definition 2.3. The sensor bundle is composed of a source quad, based on a source plane, and a set of target triangles, all based on a target plane and oriented equally. It contains all rays starting at the source quad with an endpoint in one of the target triangles.

These two bundle should be treated separately throughout the algorithm. It requires some added complexity to compute the intensity of the light hitting the sensor plane, which then results in the light intensity we need as output of our bundle tracing algorithm. However, this can be done quite efficiently by using some of the properties of a viewing projection, much like a rasterizer projects the scene onto the viewport, instead of computing the refraction of the lens.

For this project, only the rays passing through the center of the lens are traced to the sensor plane. This restricts the camera to a camera obscura. However, this can easily be extended to a more realistic camera by tracing all the rays passing through the lens at any point. This will add a depth of

field. Using only the center of the lens means the lens bundle has no source triangles, but only a single source point.

The light sources used in this project are spherical and omnidirectional. This means the bundle leading to a light source should have a spherical target. Since light sources are not physical objects, but solely emitters of light, a bundle does not need to divided along the sides of the light sources. Instead, a subbundle to the light source is created, whereas its main bundle can continue unchanged.

With the assumption that the light source emits light equally across its surface, only the information on the part of the light source that is visible is required. Such a bundle, leading to a light source, can thus be defined as the bundle from the source plane containing all the rays leading to the spherical light source. No exact definition of this bundle is given here yet, as some work on the subject is still required.

2.4 Subdivision

It is clear that a single bundle from the viewpoint is not sufficient to reach the whole scene. To be able to trace light from one plane to another or from a plane to the viewpoint, multiple confined bundles are needed, each with a target plane defined. A bundle that is part of a larger bundle, is called a *subbundle*. In short, the main unconfined bundle from the camera needs to be subdivided into confined bundles, such that these bundles combined contain all the rays of the original bundle and the original contains all the rays of its subbundles. If this is the case, then the bundles form a *complete subdivision* of the camera bundle.

To achieve this, we can start by taking the nearest triangle from the camera. Obviously, the first subbundle that is needed, is the confined bundle leading to this plane. This subbundle will contain all the rays that hit the triangle. The remaining subbundles should then cover the area around the triangle, so that, combined, they contain all the rays that do *not* hit the triangle. Since these subbundles do not lead to a target plane yet, they remain unconfined. Figure 5 shows how three bundles can be created along a triangle, subdividing the area around it.

If we can restrict a bundle to a certain area around a triangle, then we can continue to follow these new subbundles to the next nearest triangle they hit. Then we can simply repeat the same procedure using this plane. Once all of the bundles have been traced, i.e. there are no unconfined subbundles left that hit a plane, we are left with a set of confined subbundles. If the scene is closed, so that no light can escape through some opening, then these confined bundles will be a complete subdivision of the camera bundle.

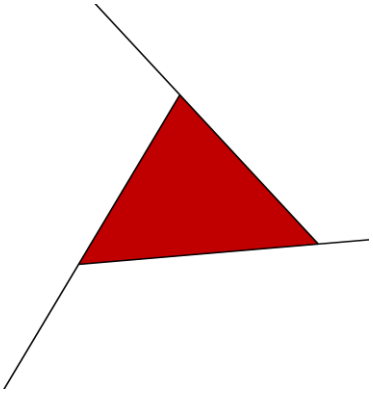

Figure 5: The general subdivision of a bundle along a triangle

2.5 Cutting edges

To restrict a bundle to only those rays passing around a part of the triangle, the edges of the triangle can be used. The rays will only be allowed to pass on one side of the edge, cutting away those rays on the other side. Then, by using two edges together, one allowing rays to pass outside the triangle and one on the inside, a usable result can be achieved (as seen in Figure 5).

The edges that are used to cut the bundle will be called *cutting edges*. A bundle containing a cutting edge consists of all the rays from each point on the source plane to each point on the target plane, where the ray passes along the positive side of the cutting edge. Geometric algebra provides a good definition of the positive side of a line in the conformal model. Of course, this side is defined only from a certain viewpoint, which will, in the case of bundles, be placed somewhere on the source plane. To construct this positive side, one might construct a plane from the viewpoint and the edge ($\Lambda \wedge p$, where p is the viewpoint and Λ the line of the cutting edge), and find the positive side of the plane. With multiple cutting edges, the bundle contains only those rays passing along the positive side of every edge. Intuitively, when looking at the triangle, it is clear which side should be positive for the bundle to pass by the triangle.

With this definition of cutting edges in a bundle, it is not immediately clear which rays from the source plane will still hit the target plane and, moreover, whether any ray still hits the target plane at all. If no rays are left that hit the target plane, the bundle is said to be *empty*. This means it is practically useless and no longer needs to be traced further, which is very valuable information. When tracing the rays inside the bundle, only those rays need to be considered that pass along the cutting edges. To have to do

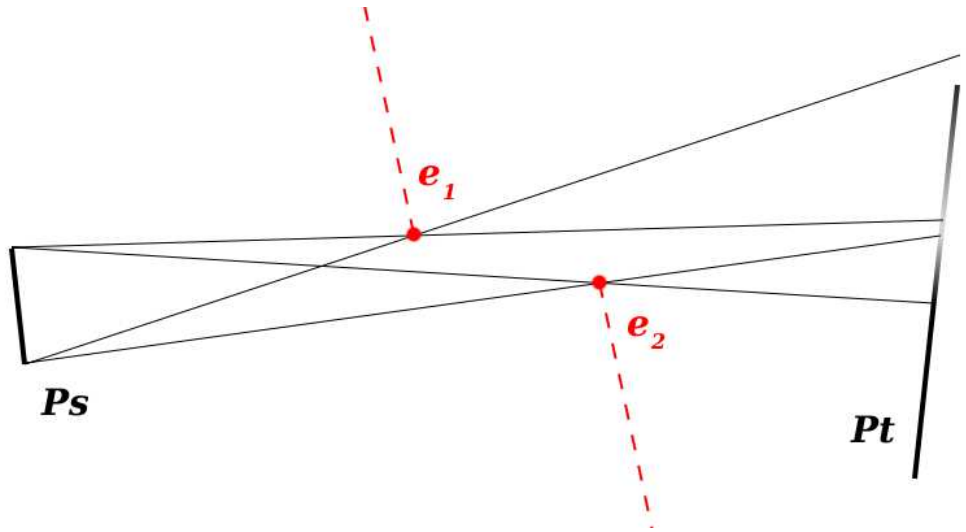

Figure 6: A bundle with source plane Ps and target plane Pt, drawn in 2D, restricted by two cutoffedges, e_1 and e_2, where the negative side of the edge is drawn as a dotted half-line.

this check for every ray for every edge would be very costly. All in all, it is important to be able to acquire this information as efficiently as possible.

Additionally, cutting edges can cause part of the target triangle to no longer receive any rays. This is clearly visible in figure 6, where the black part of the target plane lies completely in shadow. It is good practice to adjust the target plane to exclude any shadowed areas. It might even be useful to subdivide a confined bundle to separate the fully lit areas from the penumbra. This way, the bundles leading to the fully lit areas are no longer restricted by their cutting edges, which makes tracing the light through these bundles a lot easier.

Besides a few obvious special cases, where it is easy to see if a bundle is empty, a good method of finding the valid rays inside a bundle is yet to be found. For now, a simple brute force approach is still used.

Whether a bundle leading to a triangle is empty or not is important information when finding a subdivision. If the triangle is not hit by any rays, we can skip it entirely. This often raises precision problems, when a cutting edge is adjacent to one of the edges of a triangle. If the edge cuts away most of the triangle, a part at the corner may still be hit because of a precision problem. Whether to decide if a point on a triangle is still hit in such a scenario requires some added logic based on the situation.

2.6 Bouncing from planes

When a bundle becomes confined, i.e. it hits a plane, several things may happen, depending on the settings of the bundle tracer. In any case, a number of *child bundles* may be created, either through reflection or refraction. Such a child bundle contains the rays that start at the end of those of its parent, in such a way as is appropriate for the given reflection or refraction. For example, in the case of a reflection, the child bundle will contain all the rays starting at its parent's target plane, but only leading to the same side of that target plane as the side the parent bundle came from. In the case of a refraction, only the rays leading to the opposite side are contained.

Upon creation of such a child bundle, there are typically no restrictions to the bundle, which means it has no cutting edges and is unconfined. The main restriction that will be shown later, is that it only contains those rays that start from a point where at least one ray from its parent ended. These properties suggests that the restrictions of rays starting in a child bundle are inherited from its parent, but that the child bundle will therefore not inherit the structural restrictions, i.e. the cutting edges.

With this concept of child and subbundles, the light through the scene can be traced. Although the bundles provide information on the presence of light across a bundle and, after doing a little more work, across multiple bundles, they do not provide any direct information on the amount of light. The intensity of the light across a scene is determined mainly by the presence of light sources and the effects of reflection, refraction and absorption on the scene‘s planes. The next section will study the effect of reflection using bundles.

3 Reflection

This section deals with the computation of the amount of light that is reflected from a bundle onto a child bundle. The computation is limited to the amount of light in one specific reflected bundle, which is a generic confined bundle and therefore ending in another target triangle. This child bundle starts at the same triangular plane where the first bundle ended. In short, the light from a source bundle is reflected in a triangle onto a reflected bundle. In this section, the triangle from which the source bundle emerges is called the source triangle. The triangle on which the light is reflected is called the reflection triangle. The plane to which the reflected bundle leads is called the target triangle.

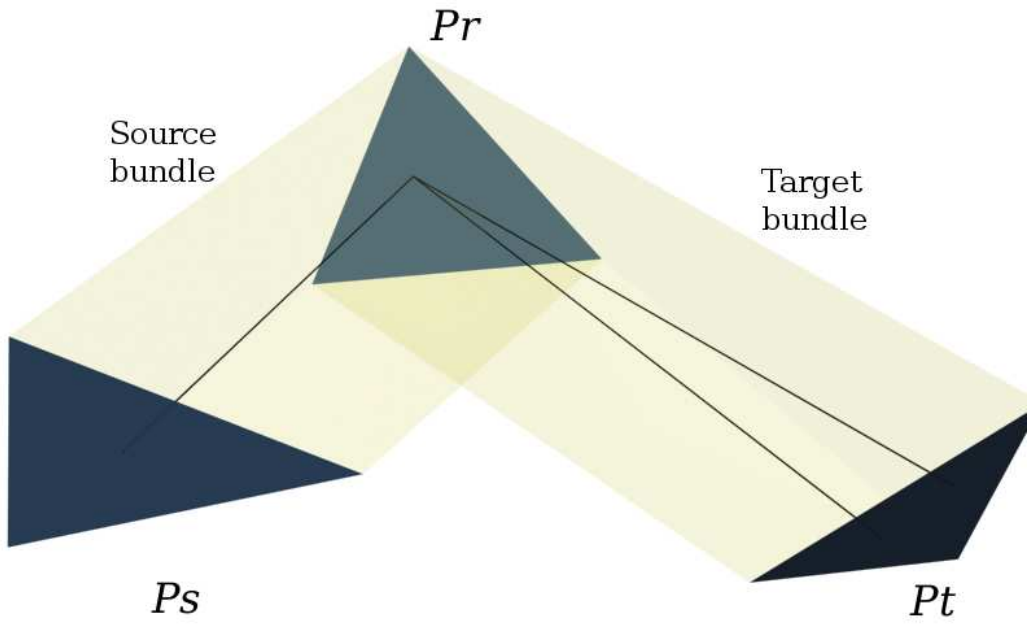

Figure 7: Two separate reflections, starting from the source triangle Ps, reflecting on the reflection triangle Pr and hitting the target triangle Pt

Perfect reflection follows the law of reflection from physics, which states that the angle of incidence must be equal to the angle of reflection, both seen relative to the normal vector of the plane. In terms of Geometric Algebra,

we can define this as $r = n\ l\ n^{-1}$, where r is the reflected vector, n is the surface normal and l is the incoming light vector. An alternative notation uses the reflection plane A directly: $r = A\ l\ A^{-1}$.

Most surfaces, however, are not perfectly flat, which causes the reflection to become diffuse. Many models exist to describe a general surface, such as that of Torrance-Sparrow (see chapter 9.4.2 of [4]). The model described here is inspired by this model.

The goal of the reflection computation is to find the distribution of the light intensity across the reflected bundle given the intensity across the source bundle. For this, a definition of intensity in a bundle is needed.

Microfacets

The modeling of rough surfaces is based on the concept of microfacets, small oriented parts of a plane with an equal length across the plane (see Figure 8). Every microfacet causes a perfect reflection, which is identified by its normal.

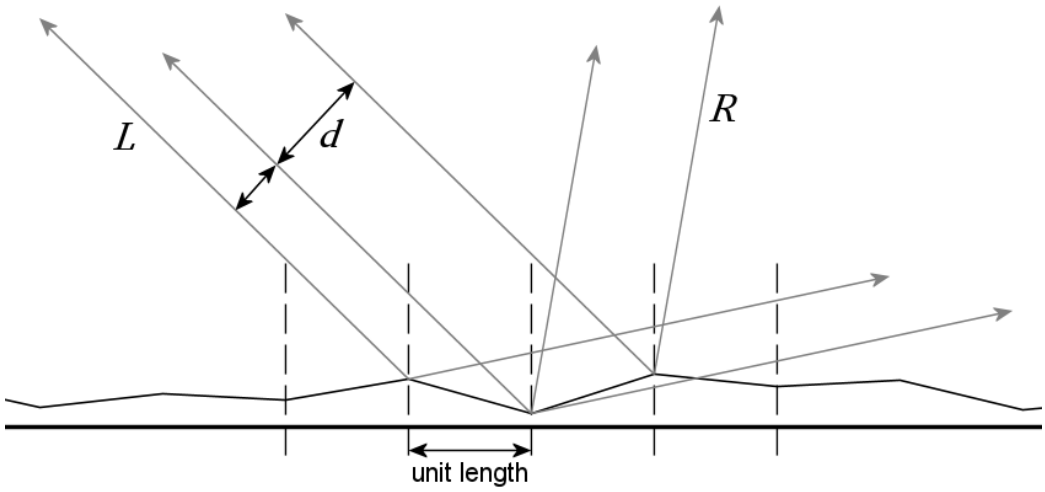

Figure 8: Several incoming light vectors (denoted L) are reflected on a distributed surface.

The length of a microfacet is considered infinitely small. Since the rays are considered to have some width, this means that every ray hitting a plane will hit an infinite amount of microfacets. The distribution of the orientation of the microfacets is a characteristic property of the plane's material and determines what will happen to the ray upon reflection or refraction.

Intensity across a bundle

The motion of light is best described as a wave motion, causing it to move from a single point along the surface of a growing sphere. Since each part of the source triangle that is used for the computation of reflection is so small, we can assume its size to be negligible compared to the distances the light travels.

With this assumption, we can say that the total intensity of the light from a very small segment of the source triangle remains equal across the surface of a growing sphere as the light moves further from the plane. This means the intensity of the light decreases by a square factor of the distance, as the radius and surface area of the sphere increase. A single ray, representing a volume of light leaving the source plane, is covered by a part of the surface of this sphere, which grows along the same factor as the entire surface. Hence, the intensity per surface area of this ray will also decrease by a square factor of the distance, the length of the ray. For more information, see chapter 5.2 on Basic Radiometry of [4].

With this we can say that the intensity of light originating from some small segment of the source triangle and traveling in some direction towards a small segment of another surface is defined at unit distance from the center point of this small segment and hitting a unit surface area on the sphere surrounding this point. For simplification of the next sections, we will speak of this unit of light as a ray, as though it were a simple line. The segment of the source triangle is simplified to a point. Bear in mind however, that this ray has a surface area on both ends, therefore describing a volume of light. It will thus retain the property of decreasing intensity by a square factor of the distance.

3.1 Reflection using integration

In this section an attempt is made to find a function describing the intensity of light contained in the reflected ray, given a set of rays from the source triangle to a point on the reflection triangle. To do this, a function describing this intensity for a single point on the source triangle is sought. Then this function is integrated along the points on the source triangle to find a general function for the combined intensity of the reflected ray, from a point on the reflection triangle to a point on the target triangle. Although the attempt to integrate was unsuccessful, this section should provide some insight in the computation of the light intensity through reflection. To keep the model simple, the computation of reflection is first done in 2D. Any triangle is thus simplified to a line segment.

In section 3.2 a similar approach is used in 3D, without the attempt at integration. This will result in a far simpler function, since common geometric properties can then be used freely.

Let us first have a look at the general idea behind the reflection model. There are four factors that determine the intensity of the reflection:

- Then intensity of the incoming light, which is linear with the intensity of the reflected light.
- The square factor of the distance the incoming light has traveled. This represents the attenuation of the light.
- The chance of hitting the right microfacet to fit the direction of reflection. Unlike the model of Oren-Nayar, where each microfacet causes a perfectly diffuse reflection, each microfacet in this model reflects perfectly specular. This means that only one orientation will lead to the required direction of reflection. Since the microfacets are so small, we can assume a distribution of microfacets to be hit by each ray. The chance of hitting the right microfacet is then determined by a distribution based on the orientation of the microfacet. A Gaussian distribution over the slope of the microfacet is used.
- The amount of light to hit the microfacet. This can be thought of as the width of the bundle of light that hits the microfacet (shown as d in image 8). This factor is equal to the cosine of the angle between the microfacet's normal and the incoming light ray.

The integration was attempted using three different representations for the rays, in order to determine the starting and ending points of the incoming ray and the reflected ray. The first used only the distance across the three triangles. The second used the angles between the endpoints of the rays and the endpoints of the triangles. After the first two failed, the last attempted to use a combination of both.

Each of these three models should have two varying parameters to determine the starting and ending points of the reflected ray. A third parameter is then used for integration, which will sum all the incoming rays.

The attempted solution using angles showed a non trivial angular relationship, between the ending point of the incoming rays and the starting point of the reflected ray, which made finding the reflection function difficult, even without attempting to integrate it. The other two approaches did result in a valid formula for a single ray, expressed using a third variable for integration, but these proved to be impossible to integrate, even in the 2D representation used here.

3.2 Reflection using a finite sum

To avoid the problem of integration entirely, this section discusses the use of a finite sum. The source, reflection and target planes are divided into small segments using a grid. Reflection computation is then done for each combination of segments on these grids. These segments are treated as points with a specified weight, the surface area of the cell. With this simplification, any kind of shading model can be used to compute the light intensity throughout the reflection. Here, the model discussed in the previous section will be adopted, with some modifications to simplify it for its new use.

The normal of the microfacet can be found by simply taking

$$\tilde{n} = \frac{L + R}{\|L + R\|}$$

where L is the normalized light vector and R the normalized reflection vector. This vector is similar to the half vector H used in the Blinn reflection model. Now the cosine of the angle between the normal of the microfacet, $\tilde{n}$, and the light vector, L, can easily be calculated by

$$\tilde{n} \cdot L$$

The slope of the microfacet is equal to the sine of the angle between $\tilde{n}$ and n, which can be computed as $\|\tilde{n} \wedge n\|$. The distribution based on this slope requires a small correction factor to keep it usable for multiple values of σ. This leads to a distribution such as

$$ND(\|\tilde{n} \wedge n\|, 0, \sigma) \cdot \frac{\sigma}{0.4}$$

Putting it all together results in

$$\tilde{n} = \frac{L + R}{\|L + R\|}$$

$$I_r = \frac{I_i \, (\tilde{n} \cdot L) \cdot SD(\|\tilde{n} \wedge n\|, 0, \sigma)}{L' \cdot L'}$$

Several graphs showing the results of this function in different situations can be seen in figure 9.

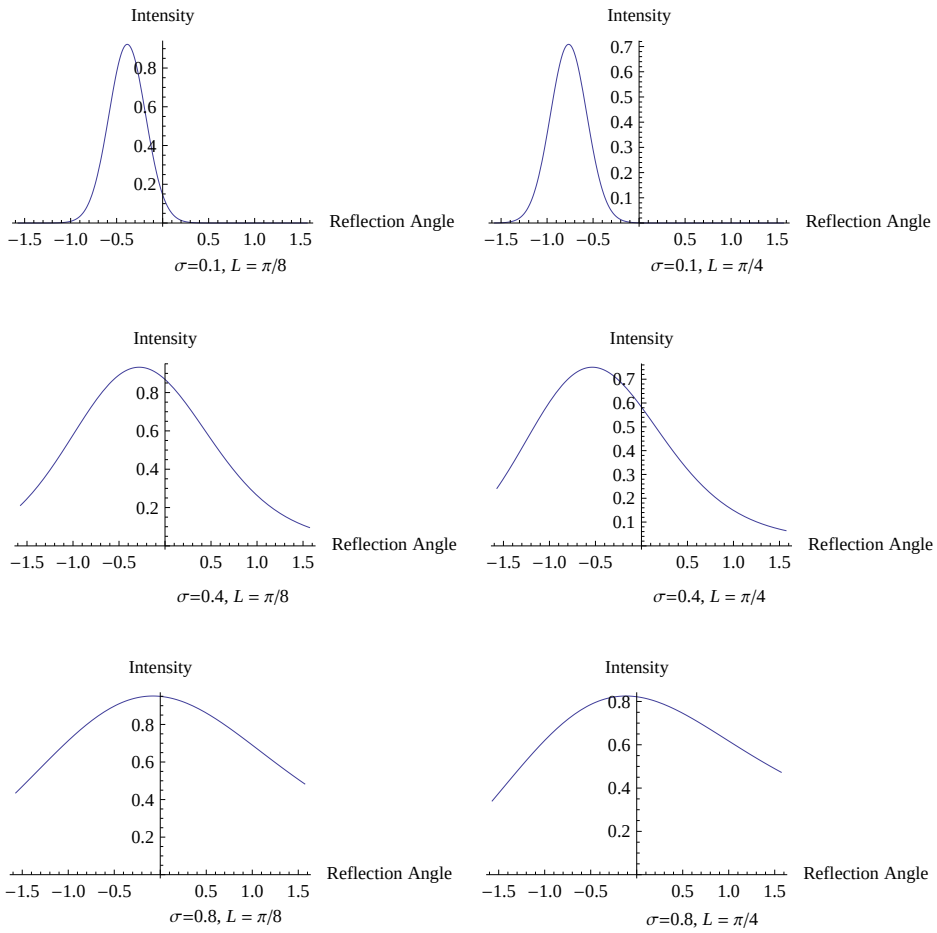

Figure 9: The resulting intensity of a reflection over the different directions of the reflection vector. L and σ are varied between the graphs.

4 The algorithm

Bundle tracing works in three steps:

- Generating the bundles using the scene's geometry.
- Computing the intensity of the light across the bundles, i.e. tracing the light through the bundles.
- Generating the image using the computed light intensities.

Before generating any bundles, it is important to know the order of the triangular faces in the scene. In this project, this is done by using a Binary Space Partitioning (BSP) tree. These trees divide the entire space into convex subspaces along the triangular faces the scene contains. At each triangular face, the scene is thus divided into half, one half on the front side of the face, one on the back side. This allows the triangles to be sorted front to back from any point of view. A small addition to a normal BSP tree is the ability to add light sources at the leaves of the tree. These are then also sorted amongst the triangles, but do not subdivide the scene. For a static scene, the BSP tree needs to be created only once.

4.1 Generating the bundles

A single unconfined bundle can be created from the lens of the camera. This bundle is then subdivided along the scene's geometry, leaving only confined bundles, to triangles and light sources. The bundles leading to triangles can then be reflected, creating a child bundle to be subdivided. The resulting confined bundles may also be reflected, up to a given limited number of bounces. This limit can be set to depend on the surface roughness, σ, such that light hitting reflective surfaces, such as mirrors, will be reflected more often. Once the limit is reached, only those subbundles to a light source need to be stored, since the others do not contribute any light.

During this stage, it is very important to know the order of the triangles as they appear for the bundle from the camera or a given reflected child bundle. This order will need to be redetermined for each (reflected) child bundle. With a BSP tree, this means the correct order of the tree needs to be found starting from one of its leaves.

The algorithm for generating the bundles with a single reflection is given in Algorithm 1.

Algorithm 1 Bundle generation

```
Input: A scene with triangles, light sources and a camera
Create bundle from the camera
for each triangle and light source in the scene {in front to back order from
the camera position} do
  for each subbundle generated before this triangle or light source do
    if the subbundle hits the triangle then
      Generate three subbundles {by cloning and adding the triangle's
      cutting edges}
      Generate a reflected bundle as a child of the camera subbundle
    end if
  end for
end for
for each reflected bundle do
  for each triangle and light source in the scene {in front to back order
  from the reflected bundle's source triangle position} do
    for each subbundle generated before this triangle or light source do
      if the subbundle hits the triangle then
        Generate three subbundles {by cloning and adding the triangle's
        cutting edges}
      end if
      if the subbundle hits the light source then
        Generate a bundle to the light source as a child of the reflected
        bundle's parent
      end if
    end for
  end for
end for
Output: A set of bundles
```

4.2 Computing the intensity

Once the bundles have been generated, the intensity across the bundles is computed. This is done by subdividing the triangles on each bundle's source and target plane along a grid, aligned to the edges of the triangle, where the number of cells is chosen to meet a given accuracy. Naturally, the grid on a bundle's target plane should be the same as on its child's source plane. The accuracy should depend on the direct visibility of the light across the bundle. In short, it should decrease as the bundle is the result of multiple bounces. The resolution of the cells on the sensor plane should equal the desired resolution of the output image.

Then, starting from the sensor plane, each cell is traced to each cell of the subbundles' target planes. In this stage, the resolution of the grid at the subbundles' target planes can be determined. From there, the intensity is found by stepping through the child bundles until a light is found. At each target plane cell, the intensity of the light from the child bundles that is reflected back to the source plane cell is added, as described in section 3. Of course, at each bundle, only the light passing along the cutting edges is added. The computed values are stored in an intermediate image.

The algorithm for tracing the light through the bundles with a single reflection is given in Algorithm 2.

Algorithm 2 Bundle tracing

```
Input: A set of bundles and a black intermediate image
for each bundle b generated from the camera do
  for each pair of points in the grids of the source and target triangles of
  b do
    for each child bundle of b leading to a light source do
      Add the intensity of the reflected light to the image corresponding
      to the sensor plane
    end for
  end for
end for
Output: An intermediate image filled with light intensity values
```

4.3 Generating the image

Once the light intensity hitting the sensor plane is found, this intensity is mapped to a color value, so that the monitor will display the intensity correctly. To achieve this, gamma correction is performed:

$$I_c = I_l^{1/\gamma}$$

where I_c is the intensity of the color value, ranging from 0 to 1, I_l is the intensity of the light and gamma is constant factor depending on the monitor and system settings. A value of 2.2 is common, following the sRGB standard.

This final step is important for the realism of any graphics implementation. A good lighting method, including both a shading function and a lighting algorithm, can only be properly assessed if the result is properly displayed.

This mapping finally results in an image of the scene.

5 Related and future work

Bundle tracing borrows heavily from other ray tracing methods and from rasterization. In terms of quality, it should be equal to other ray tracing methods, since bundle tracing does not restrict the manner in which rays are created throughout a scene. Rasterization is clearly inferior in quality however, as it is heavily restricted by rendering passes and per vertex or per pixel operations. In terms of speed, bundle tracing seeks out the middle ground between these two. Rasterization is likely the fastest rendering algorithm available, while ray tracing can take hours to generate a single image. Because of its geometric per bundle basis, bundle tracing should always be faster than ray tracing, especially in less complex scenes.

Similar attempts to accelerate ray tracing by using volumetric primitives have been done. Beam tracing among them is the most similar to Bundle Tracing, however, the techniques are fundamentally different. While beam tracing uses pyramidal beams with a single source point, bundle tracing uses a more free primitive with a source triangle. While this adds to the complexity of the computation, it also makes Bundle Tracing more versatile. Beam tracing requires a new beam to be traced for every reflection for every drawn point in the scene. Bundle tracing does not require this, where only one bundle per reflection per triangle is needed.

Bundle tracing should scale well for use in Global Illumination, where secondary or tertiary reflections commonly require less geometric complexity, thus allowing bundle tracing to benefit more from its triangular nature. It is also very useful for rendering scenes with soft shadows. The bundles can be split along the penumbra's edges, thus requiring minimal effort for a geometrically correct result. Another possible advantage of Bundle Tracing is the computation of the depth of field, which comes naturally with the bundle representation.

Like Beam Tracing, Bundle Tracing can fully utilize anti-aliasing along the edges of every triangle in the scene, since a similar kind of subdivision is done. In the case of Bundle Tracing, this can even be combined with the depth of field by repeatedly drawing the computed shape of the aperture instead of a single pixel. For beam tracing this can not be done entirely correctly without using more beams, since the geometric information of the scene originates from only a single point, thus occluding parts of the scene around the edges of objects (see Figure 1 for an example). This kind of anti-aliasing is an improved way of sampling used in ray tracers, made possible by the geometric information represented by the bundles. A complete proof of concept and implementation of this kind of sampling still remains to be made.

While many improvements could easily increase the quality of the output, improvements regarding efficiency are also still required. The most important of these is determining whether a triangle is hit by a bundle, but also an improved way of finding out exactly which rays of a bundle hit a triangle is important. Once a good method for this has been found, an implementation of bundle tracing may attempt to compete with current ray tracing implementations.

6 Current results

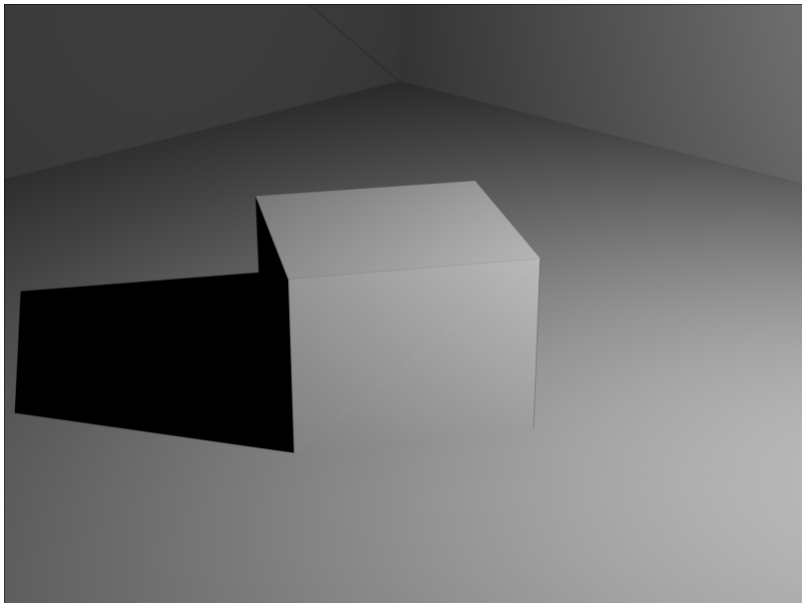

Figure 10: A render of a very basic scene.

The images presented here are rendered using my first implementation of the algorithm. Many common features of ray tracers and rasterizers are not implemented here and it is only run on a very simple scene showing a box inside another box.

Figure 10 shows the result of rendering this very basic scene. Notice how the edges of the box are smooth and the shading is comparable to that of other modern techniques.

Figure 11 shows the same scene with a soft shadow computation. Although some areas are still incorrectly shadowed here, it already provides a nice view of future possibilities.

These images do not show the limitations of the algorithm presented. The lines that are visible between the triangles of the scene, the simplicity of the scene and lighting and the errors in the shadowing are caused by a first and very basic implementation of the bundle generation algorithm. The main problem these images present lies in the tracing of light through these bundles.

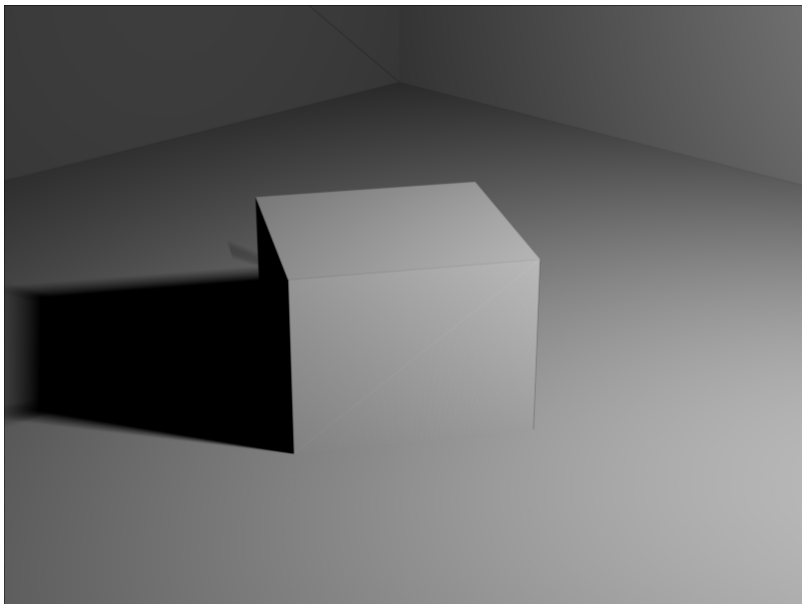

Figure 11: A render of a very basic scene.

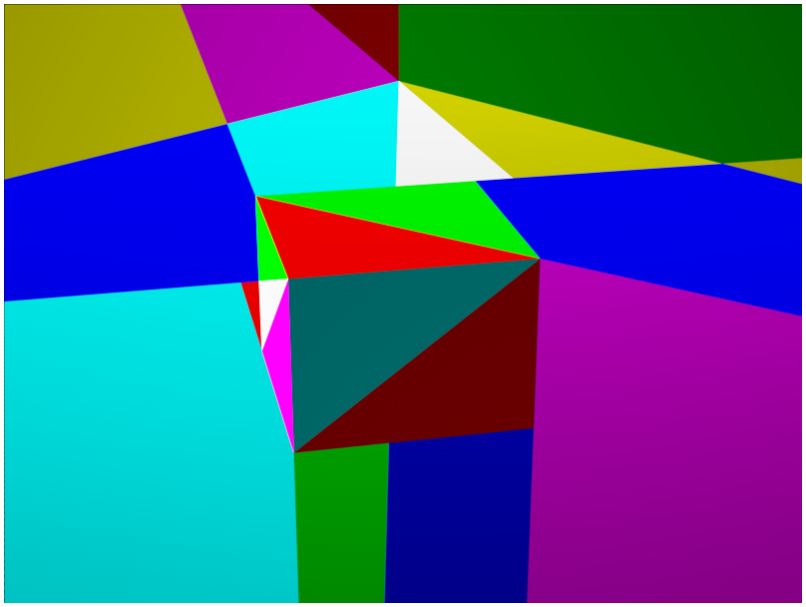

Figure 12: The same scene as in Figure 12. The contributed light of each separate bundle is given a different color.

References

[1] Frank L. Pedrotti, Leno S. Pedrotti and Leno M. Pedrotti,
Introduction to Optics, third edition, Pearson, 2007

[2] Leo Dorst, Daniel Fontijne, Stephen Mann,
Geometric Algebra for Computer Science, Morgan Kaufmann, 2007

[3] F. S. Hill, Jr. and Stephen M. Kelley, Jr.
Computer Graphics Using OpenGL, third edition, Pearson, 2007

[4] Matt Pharr and Greg Humphreys
Physically Based Rendering, first edition, Elsevier, 2004